Configuration Manual

MSc Research Project
Masters in Cloud Computing

Rishabh Sinha
Student ID: X21171203

School of Computing
National College of Ireland

Supervisor: Dr Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rishabh Sinha
Student ID:	X21171203
Programme:	Masters in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr Aqeel Kazmi
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	rishabhsinha
Date:	13th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rishabh Sinha
X21171203

1 Predictive Phase- Workload CPU and Memory Prediction based on Historical Data

Experimentation on the dataset from Bitbrains IT Services Inc. to assess predictive algorithms using historical workload data and predict CPU and Memory Usage of a workload. Shen et al. (2015)

1.1 Prerequisites

- Google Collab with Python3 Runtime Type and T4 GPU accelerator.
- Create and activate a python virtual environment by running below command (**Optional**, **if running on VS Code in local**). Execute "*python -m venv predenv*" and "*source predenv/bin/activate*"
- Install all the Python packages from '*predrequirements.txt*' for this phase of research. Execute "*pip install -r predrequirements.txt*" .Figure 2

1.2 Steps to perform

- Step 1: Go to "`https://colab.research.google.com/`".
- Step 2: Go to File and click on Upload notebook.Figure 1

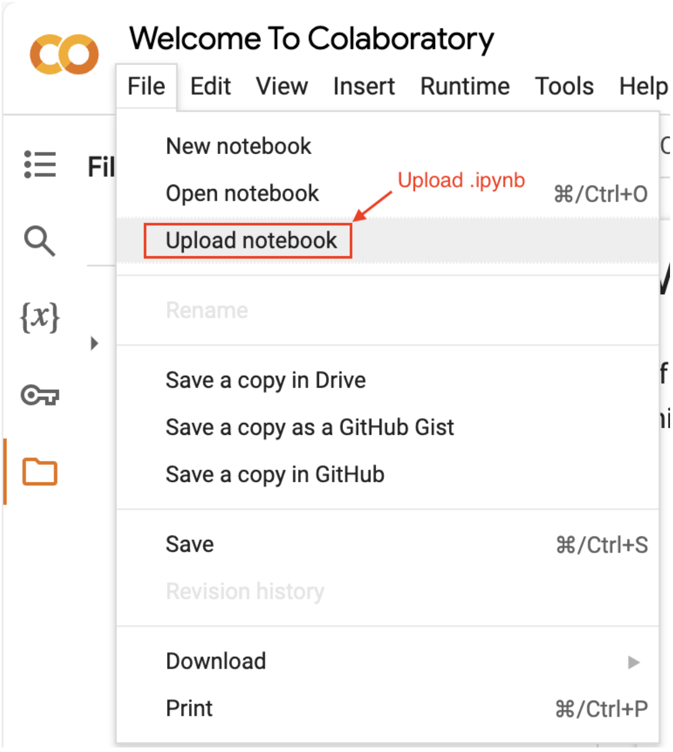

Figure 1: Upload 'MultiCloudPredictionPastData.ipynb' file to Google Collab.

- Step 3: Select "*MultiCloudPredictionPastData.ipynb*" file from */ICT Solution/x21171203-Research-CodeArtifact-main/WorkloadPerformancePrediction* directory as shown in Figure 2 directory strcuture.

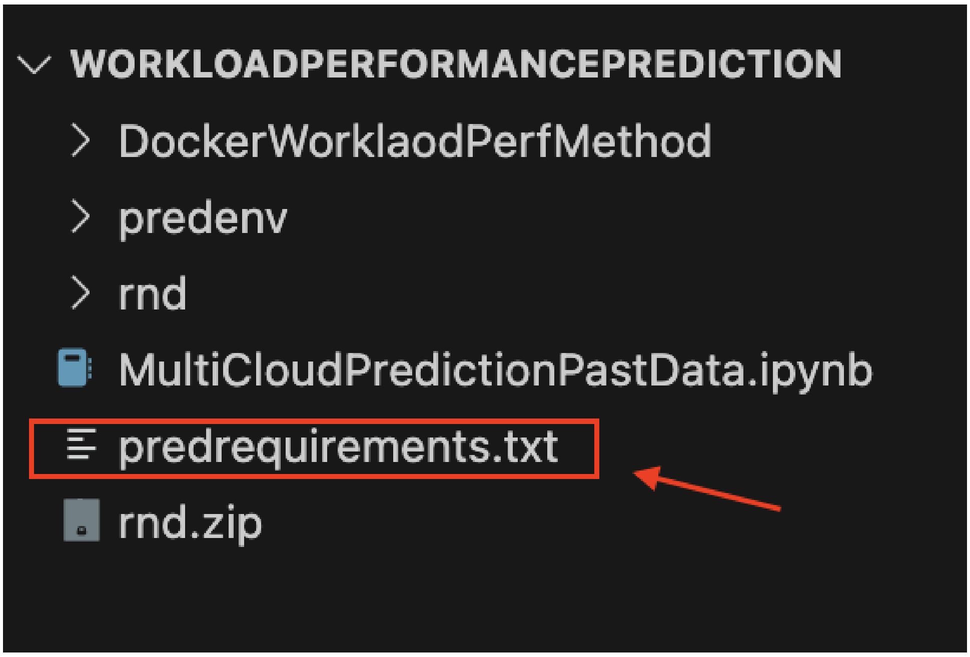

Figure 2: Requirements file for the predictive phase.

- Step 4: Run all the cells in the uploaded Python Notebook to reproduce Experiments.

- Step 5: The cell 2 & 3 in python notebook downloads the *rnd.zip* file from "`http://gwa.ewi.tudelft.nl/fileadmin/pds/trace-archives/grid-workloads-archive/datasets/gwa-t-12/rnd.zip`" and unzips it. Use "*curl -O*" if your OS does not support "*wget*" command. After the completion of this step 'rnd' folder and 'rnd.zip' file will be available in the Root working Directory where the python notebook in executed as can be seen in Figure 2 .

- Step 6: Run all the consecutive cells in order to reproduce all the steps and analysis on different predictive algorithms on the Bitbrains IT Services Inc. dataset. Shen et al. (2015).

The above mentioned steps enable to predict CPU and Memory Usage requirement of a workload based on time series historical data available from BitBrain IT Service Inc. dataset. This phase compares different predictive model in the "*MultiCloudPrediction-PastData.ipynb*" python notebook file. This aims in selecting best predictive model that can perform better in a given scenario of predicting CPU and Memory usage of a workload. This prediction gives an estimate of workloads configuration requirements(CPU, RAM) for better performance in any computational environment.

2 Search phase: Azure & AWS Data Analysis

- Step 1: */ICT Solution/x21171203-Research-CodeArtifact-main/MultiCloudDataAnalysis/* directory.

- Step 2: There are three .ipynb files in the directory as mentioned in Figure 3. These files can be executed cell by cell in Google collab. These .ipynb files contain a thorough analysis of various SKU and configuration parameters offered by Azure and AWS.

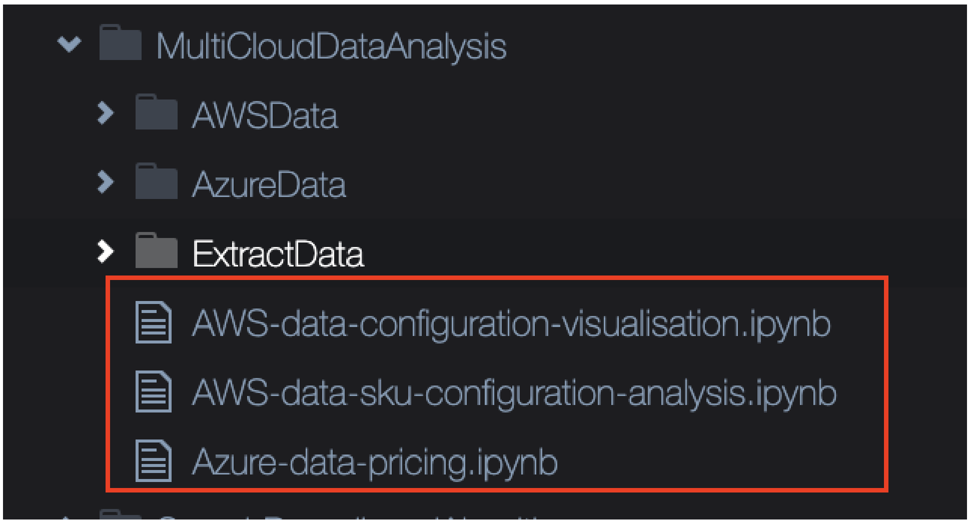

Figure 3: Azure & AWS SKU data analysis ipynb files

Note: It is recommended to use AWS Cloud9 for easy interaction with the AWS Bulk price API.

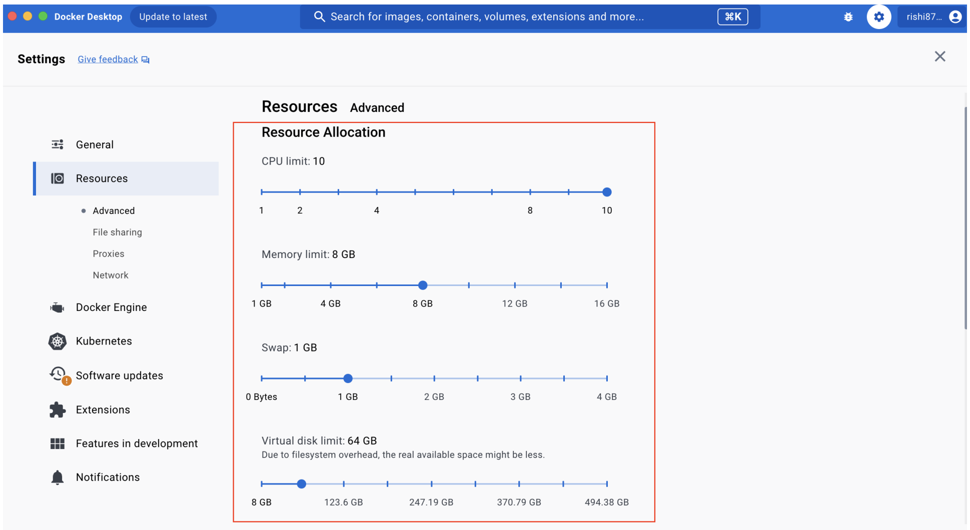

Figure 4: Resources allocated to Docker Engine for new workload execution

3 Predictive Phase- Docker Profiling of a New Workload

3.1 Prerequisites

- Install the Docker Desktop app from the Docker official website. `https://www.docker.com/products/docker-desktop/`. The authors docker environment is set as mentioned in Figure 4
- Create and activate a python virtual environment by running the below command. Execute "*python -m venv dockenv*" and "*source dockenv/bin/activate*"
- Install all the Python packages from '*dockrequirements.txt*' for this phase of research. Execute "*pip install -r dockrequirements.txt*" to install all the required packages.Figure 2

3.2 Steps to perform

- Step1: Open Docker Desktop App and start Docker Engine. It should show the running status of the docker engine as mentioned in Figure 5.
- Step2: Open terminal.
- Go to "*/ICT Solution/x21171203-Research-CodeArtifact-main /WorkloadPerformancePrediction/DockerWorklaodPerfMethod*" directory. The directory structure should be as mentioned in Figure 6.
- Step3: Define workload parameters in *random_worklaod.py* main method to set the configuration parameters as mentioned in Figure 7.
- Step4: Run the docker build command to start building the docker image. "*docker build -t custom-workload-container .*" . This will execute the command as in Figure 9 and a new docker image will be built and visible on Docker Desktop as shown in Figure 8

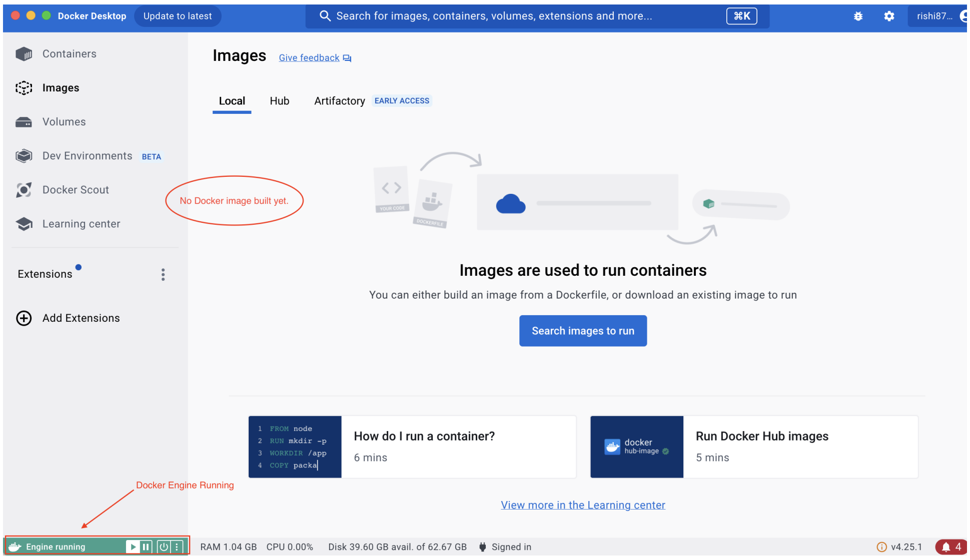

Figure 5: Docker Engine Running In Docker Desktop App

```
drwxrwxr-x@ 7 rishabhsinha  staff   224 13 Dec 16:00 .
drwxrwxr-x@ 9 rishabhsinha  staff   288 13 Dec 16:00 ..
-rw-r--r--@ 1 rishabhsinha  staff  6148 13 Dec 16:00 .DS_Store
-rw-rw-r--@ 1 rishabhsinha  staff   547  9 Dec 20:21 Dockerfile
drwxr-xr-x  6 rishabhsinha  staff   192 13 Dec 15:39 dockenv
-rw-rw-r--@ 1 rishabhsinha  staff  5886 13 Dec 15:59 random_workload.py
-rw-rw-r--@ 1 rishabhsinha  staff  1520  9 Dec 20:21 readme
```

Figure 6: Docker Simulation Directory Structure

```
def main():
    # Define workload parameters
    num_tasks = 1000
    instructions_per_task = 50
    parallel_percentage = 0.2  # 20% parallel, 80% serial
    num_threads = 2
    num_processes = 2
    num_iterations = 10
```

Figure 7: *random_worklaod.py* input workload parameters

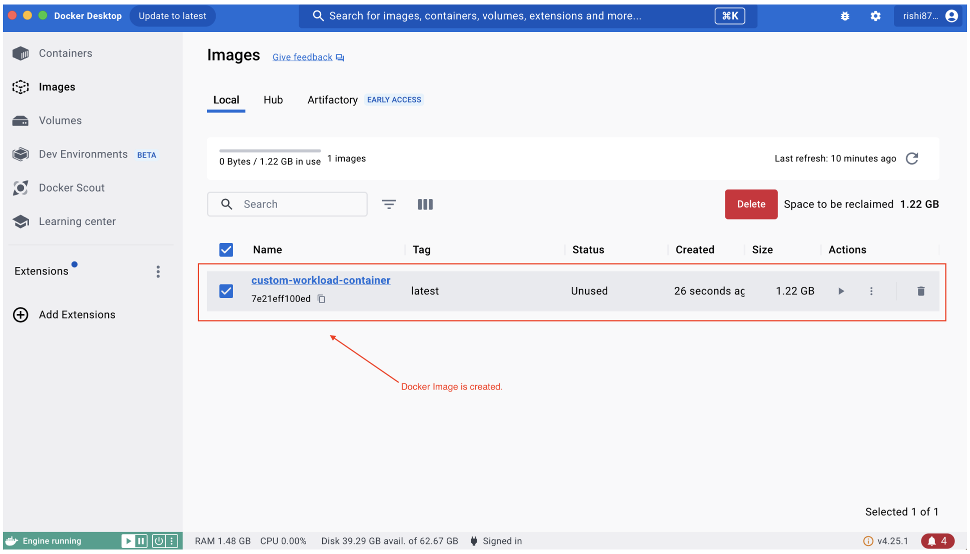

Figure 8: custom docker image built on Docker

```
(dockenv) rishabhsinha@Rishabhs-MacBook-Pro DockerWorklaodPerfMethod % docker build -t custom-workload-container .
[+] Building 0.6s (10/10) FINISHED                                                            docker:desktop-linux
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 692B                                                                           0.0s
 => [internal] load metadata for docker.io/library/openjdk:11-jre-slim                                         0.5s
 => [1/5] FROM docker.io/library/openjdk:11-jre-slim@sha256:93af7df2308c5141a751c4830e6b6c5717db102b3b31f012ea29d842  0.0s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 238B                                                                              0.0s
 => CACHED [2/5] WORKDIR /app                                                                                  0.0s
 => CACHED [3/5] RUN apt-get update && apt-get install -y gcc python3-dev python3-pip openjdk-11-jre            0.0s
 => CACHED [4/5] COPY random_workload.py .                                                                     0.0s
 => CACHED [5/5] RUN pip3 install psutil numpy joblib memory_profiler tqdm scikit-learn                        0.0s
 => exporting to image                                                                                         0.0s
 => => exporting layers                                                                                        0.0s
 => => writing image sha256:7e21eff100ed83170d0ae09bda9f41ab2ef5c0a52633d7f1e9c84687353305f2                   0.0s
 => => naming to docker.io/library/custom-workload-container                                                   0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Figure 9: Docker build command Execution

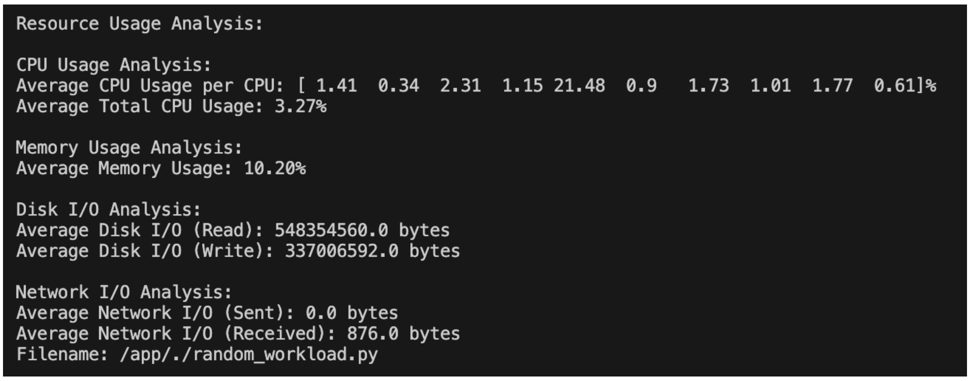

Figure 10: Docker workload execution Result

- Step5: Run "*docker run -it –rm custom-workload-container*" command to run the image in the docker container and give the CPU and Memory requirements of a workload as in Figure 10

The Mentioned steps execute a given workload on the Docker engine based on its input characteristics as mentioned in Figure 7 to estimate the CPU and memory Usage of a workload.

4 Search Phase - Selecting Optimal Cloud Configuration Parameter in Multi-Cloud environment

4.1 Prerequisites

AWS Cloud 9 (with any ec2 instance type backing it up) Python 3.8.16 version

4.2 Steps to perform

- Step 1: Go to /ICT Solution/x21171203-Research-CodeArtifact-main folder directory.

- Step 2: Create and activate a python virtual environment by running the below command. Execute "*python3 -m venv env*" and "*source env/bin/activate*" .

- Step 3: Install all the Python packages from '*requirements.txt*' for this phase of research. Execute "*pip3 install -r requirements.txt*" to install all the required packages.

- Step 4: Set the *config_file.json* with the algorithm to to use for the searching. As per the evaluation of the research project Brute Search performs better for selecting the optimal configuration parameter and cloud service provider for the small application components "Brute Search" provides optimal solution in less time. Brtue Search can be set to "True" or "False", and for another algorithm, optimum parameters are searched for the searching as mentioned in Figure 11.

```json
{
    "Data Extraction (Always / onceAday / Never)": "Never",
    "boto3 (enable / disable)": "disable",
    "Provider (AWS / Azure / Hybrid)": "Hybrid",
    "Brute Force": "False",
    "Time per region": 0.2,
    "Candidate list size": 100,
    "Proportion amount node/sons": 0.0005,
    "Verbose": "True"
}
```

Figure 11: config file .json

- Step 5: Set the *input_appcluster.json* required configuration parameters for the application components of the workload in the format mentioned in the Figure 12. additional parameters can be set based on the specific requirement such as the type of 'OS', 'type' or the 'region' required for searching the application component requirements.

```json
{
  "workload": {
    "components": [
      {
        "name": "web_server",
        "resource_requirements": {
          "cpu_cores": 2,
          "memory_gb": 4
        }
      },
      {
        "name": "database_server",
        "resource_requirements": {
          "cpu_cores": 4,
          "memory_gb": 8
        }
      },
      {
        "name": "cache_server",
        "resource_requirements": {
          "cpu_cores": 1,
          "memory_gb": 2
        }
      },
      {
        "name": "worker_node_1",
        "resource_requirements": {
          "cpu_cores": 2,
          "memory_gb": 4
        }
      },
      {
        "name": "worker_node_2",
```

Figure 12: input_appcluster.json

- Step6: Run command "*python3 MultiCloudSearch.py*" to start the searching and provide the result as shown in Figure 13

```
AWSReservedSSO_MSCCLOUD_554ef120b0d7b74a:~/environment/researchprojectsearch (main) $ python3 --version
Python 3.8.16
AWSReservedSSO_MSCCLOUD_554ef120b0d7b74a:~/environment/researchprojectsearch (main) $ python3 MultiCloudSearch.py
calculating best configuration
Searching in region hybrid
running optimizer of region:  hybrid
Node.__init__;          hash: 2387360268884754497          , depth: 0          , total_score: 0.6272
Node.__init__;          hash: 8389923730362883649          , depth: 2          , total_score: 0.987
Node.__init__;          hash: -7253668256622775l7          , depth: 1          , total_score: 0.917
Node.__init__;          hash: 8514539225192599419          , depth: 1          , total_score: 0.739
Node.__init__;          hash: -8332813301103933987          , depth: 1          , total_score: 0.8052
```

Figure 13: MultiCloudSearch.py execution output

- Step7: Output of the search is stored in "*appcluster_suggested_sku_results.json*" file. The output suggests the cloud service provider where each of the application components can be deployed as mentioned in Figure 14


```
{
    "price": 0.6272,
    "EC2 Type": "onDemand",
    "region": "hybrid",
    "instances": [
        {
            "typeName": "D4a v4",
            "cpu": 4,
            "memory": 16,
            "storage": 100,
            "region": "Central India",
            "os": "CentOS or Ubuntu Linux",
            "family": "General purpose",
            "onDemandPrice": 0.123,
            "spot_price": 0.0123,
            "discount": 0,
            "CloudProvider": "Azure",
            "components": [
                {
                    "appName": "App1",
                    "componentName": "Comp1"
                }
            ]
        },
        {
            "onDemandPrice": 0.0112,
            "region": "ap-south-1",
            "cpu": "2",
            "ebsOnly": true,
            "family": "General purpose",
            "memory": "2",
            "network": "Up to 5 Gigabit",
            "os": "Linux",
            "typeMajor": "t4g",
            "typeMinor": "small",
            "storage": "EBS only",
            "typeName": "t4g.small",
            "physicalProcessor": "AWS Graviton2 Processor",
            "processorArchitecture": "64-bit",
            "Architecture": "arm64",
            "discount": 0,
            "CloudProvider": "AWS",
            "interruption_frequency": "<5%",
```


Figure 14: appcluster_suggested_sku_results.json file

5 Running Experimentation

Step 1: Go to /ICT Solution/x21171203-Research-CodeArtifact-main/SearchBasedLocalAlgorithm folder directory. Step 2: Run below command to execute the experimentation in iteration. *python3 serial run.py* that will run the experimentation that was performed in the evaluation phase of the research.Figure 15

Figure 15: appcluster_suggested_sku_results.json file

Step 3: The experimentation folder will be created with the result output of the experimentation. the directory structure of the evaluated experimentation is visible as mentioned in Figure 16

Figure 16:

References

Shen, S., Van Beek, V. and Iosup, A. (2015). Statistical characterization of business-critical workloads hosted in cloud datacenters, *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, IEEE, pp. 465–474.